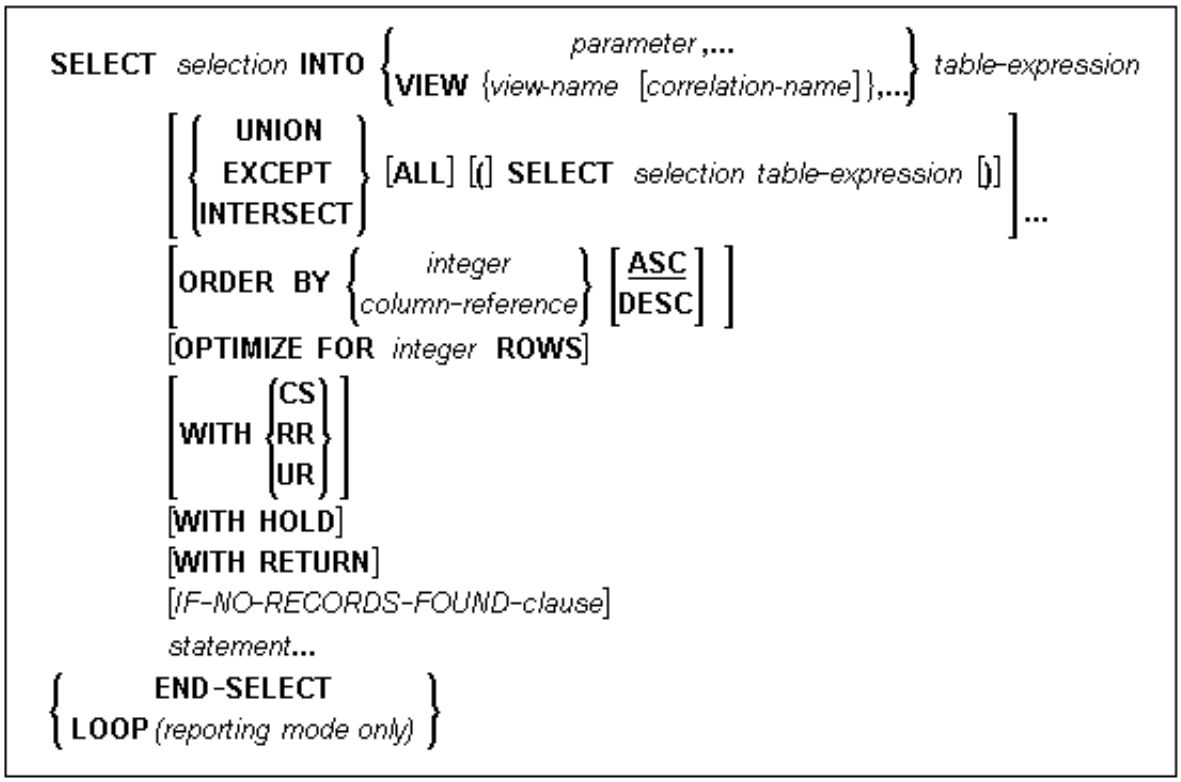
SELECT

According to the standard SQL functionality, Natural supports both the cursor-oriented selection that is used to retrieve an arbitrary number of rows and the non-cursor selection (singleton SELECT) that retrieves at most one single row.

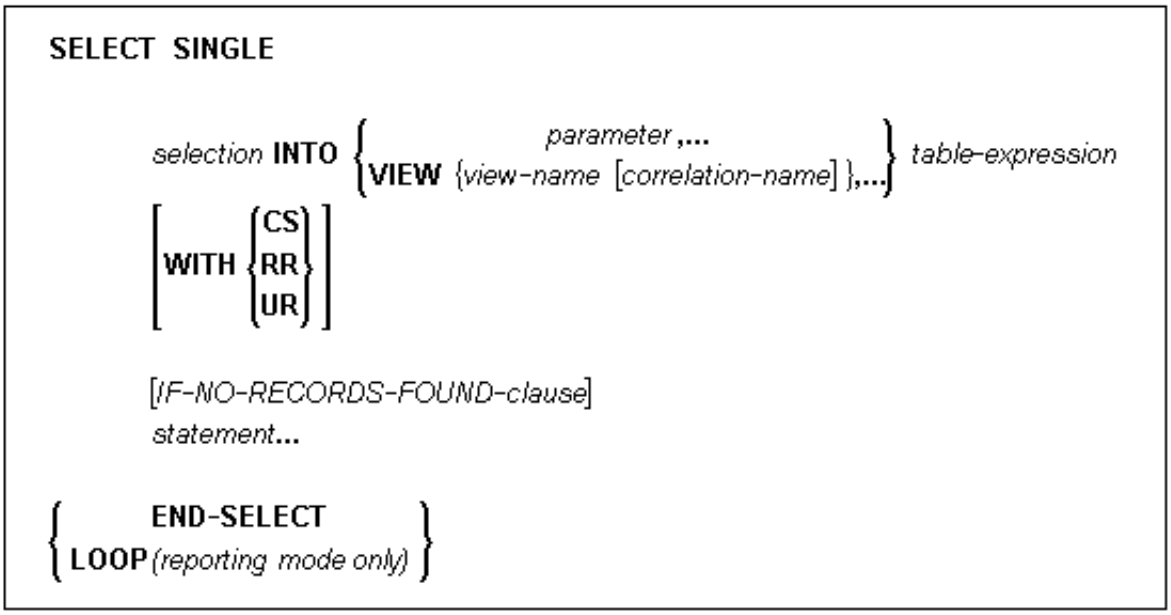
With the "SELECT ... END-SELECT" construction, Natural uses the same database loop processing as with the FIND statement.

Cursor-Oriented Selection



Like the FIND statement, a cursor-oriented selection is used to select a set of rows (records) from one or more database tables, based on a search criterion. In addition, no cursor management is required from the application program; it is automatically handled by Natural.

Non-Cursor Selection



The SELECT SINGLE statement supports the functionality of a non-cursor selection (singleton SELECT); that is, a select expression that retrieves at most one row without using a cursor. It cannot be referenced by a positioned UPDATE or DELETE statement.

---

table-expression

The *table-expression* consists of a FROM clause and an optional WHERE clause. The GROUP BY and HAVING clauses are not permitted.

Example 1:

```
DEFINE DATA LOCAL
01 #NAME      (A20)
01 #FIRSTNAME (A15)
01 #AGE       (I2)
...
END-DEFINE
...
SELECT NAME, FIRSTNAME, AGE
  INTO #NAME, #FIRSTNAME, #AGE
  FROM SQL-PERSONNEL
    WHERE NAME IS NOT NULL
      AND AGE > 20
...
  DISPLAY #NAME #FIRSTNAME #AGE
END-SELECT
...
END
```

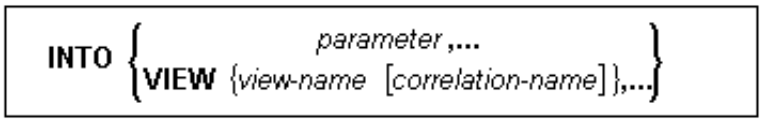
Example 2:


```
DEFINE DATA LOCAL
01 #COUNT    (I4)
...
END-DEFINE
...
SELECT SINGLE COUNT(*) INTO #COUNT FROM SQL-PERSONNEL
...
```

See further information on selection and table-expression.

Note:
In the following, the term "SELECT statement" is used as a synonym for the whole query-expression consisting of multiple select expressions concatenated with UNION operations.

INTO Clause



The INTO clause is used to specify the target fields in the program which are to be filled with the result of the selection. The INTO clause can specify either single *parameters* or one or more views as defined in the DEFINE DATA statement.

All target field values can come either from a single table or from more than one table as a result of a join operation (see also the section Join Queries).

Note:
In standard SQL syntax, an INTO clause is only used in non-cursor select operations (singleton SELECT) and can be specified only if a single row is to be selected. In Natural, however, the INTO clause is used for both cursor-oriented and non-cursor select operations.

The *selection* can also merely consist of an asterisk (*). In a standard select expression, this is a shorthand for a list of all column names in the table(s) specified in the FROM clause. In the Natural SELECT statement, however, the same syntactical item "SELECT *" has a different semantic meaning: all the items listed in the INTO clause are also used in the selection. Their names must correspond to names of existing database columns.

Examples:

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 AGE
END-DEFINE
...
SELECT *
  INTO NAME, AGE
```

```
...
SELECT *
  INTO VIEW PERS
```

These examples are equivalent to the following ones:

```
...
SELECT NAME, AGE
  INTO NAME, AGE
```

```
...
SELECT NAME, AGE
  INTO VIEW PERS
```


parameter

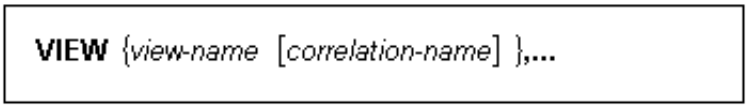
If single parameters are specified as target fields, their number and formats must correspond to the number and formats of the *columns* and/or *scalar-expressions* specified in the corresponding selection as described above (see details on Scalar Expressions).

Example:

```
DEFINE DATA LOCAL
01 #NAME   (A20)
01 #AGE    (I2)
END-DEFINE
...
SELECT NAME, AGE
  INTO #NAME, #AGE
  FROM SQL-PERSONNEL
...
```

The target fields #NAME and #AGE, which are Natural program variables, receive the contents of the table columns NAME and AGE.

VIEW Clause



If one or more views are referenced in the INTO clause, the number of items specified in the *selection* must correspond to the number of fields defined in the view(s) (not counting group fields, redefining fields and indicator fields).

Note:
Both the Natural target fields and the table columns must be defined in a Natural DDM. Their names, however, can be different, since assignment is made according to their sequence.

Example of INTO Clause with View:

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 AGE
END-DEFINE
...
SELECT FIRSTNAME, AGE
  INTO VIEW PERS
  FROM SQL-PERSONNEL
...
```

The target fields NAME and AGE, which are part of a Natural view, receive the contents of the table columns FIRSTNAME and AGE.

correlation-name

If the VIEW clause is used within a "SELECT *" construction where multiple tables are to be joined, *correlation-names* are required if the specified view contains fields that reference columns which exist in more than one of these tables. In order to know which column to select, all these columns are qualified by the specified *correlation-name* at generation of the selection list. The *correlation-name* assigned to a view must correspond to one of the *correlation-names* used to qualify the tables to be joined. See also the section Join Queries.

Example:

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 FIRST-NAME
  02 AGE
END-DEFINE
...
SELECT *
  INTO VIEW PERS A
  FROM SQL-PERSONNEL A, SQL-PERSONNEL B
...
```

Query involving UNION

UNION unites the results of two or more *select-expressions*. The columns specified in the individual *select-expressions* must be UNION-compatible; that is, matching in number, type and format.

Redundant duplicate rows are always eliminated from the result of a UNION unless the UNION operator explicitly includes the ALL qualifier. With UNION, however, there is no explicit DISTINCT option as an alternative to ALL.

Example:

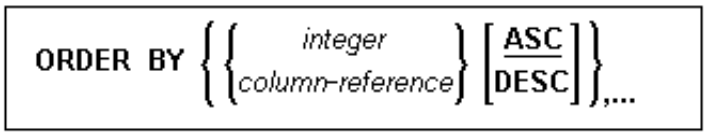
```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 AGE
  02 ADDRESS (1:6)
END-DEFINE
...
SELECT NAME, AGE, ADDRESS
  INTO VIEW PERS
  FROM SQL-PERSONNEL
  WHERE AGE > 55
UNION ALL
SELECT NAME, AGE, ADDRESS
  FROM SQL-EMPLOYEES
  WHERE PERSNR < 100
ORDER BY NAME
...
END-SELECT
...
```

In general, any number of *select expressions* can be concatenated with UNION.

The INTO clause must be specified with the first *select-expression* only.

Any ORDER BY clause must appear after the final *select-expression*; the ordering columns must be identified by number, not by name.

ORDER BY Clause



The ORDER BY clause arranges the result of a SELECT statement in a particular sequence.

Each ORDER BY clause must specify a column of the result table. In most ORDER BY clauses a column can be identified either by *column-reference* (that is, by an optionally qualified column name) or by column number. In a query involving UNION, a column must be identified by column number. The column number is the ordinal left-to-right position of a column within the *selection*, which means it is an *integer* value. This feature makes it possible to order a result on the basis of a computed column which does not have a name.

Example:

```
DEFINE DATA LOCAL
1 #NAME          (A20)
1 #YEARS-TO-WORK (I2)
END-DEFINE
...
SELECT NAME , 65 - AGE
  INTO #NAME, #YEARS-TO-WORK
  FROM SQL-PERSONNEL
  ORDER BY 2
  ...
```

The order specified in the ORDER BY clause can be either ascending (ASC) or descending (DESC). ASC is the default.

Example:

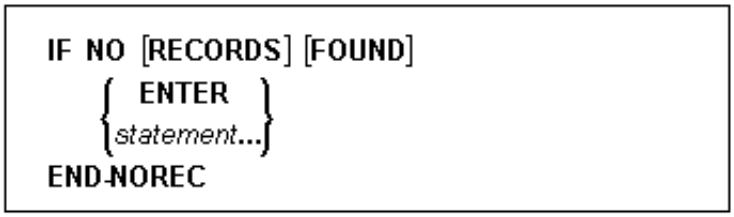
```
DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
1 NAME
1 AGE
1 ADDRESS   (1:6)
END-DEFINE
...
SELECT NAME, AGE, ADDRESS
  INTO VIEW PERS
  FROM SQL-PERSONNEL
  WHERE AGE = 55
  ORDER BY NAME DESC
  ...
```

See further information on integer values and column-reference in the SQL Statements overview page.

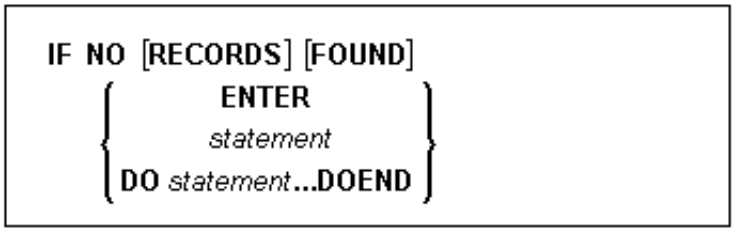
IF NO RECORDS FOUND-clause

Note:
This clause actually does not belong to Natural SQL; it represents Natural functionality which has been made available to SQL loop processing.

Structured Mode Syntax



Reporting Mode Syntax



The IF NO RECORDS FOUND clause is used to initiate a processing loop if no records meet the selection criteria specified in the preceding SELECT statement.

If no records meet the specified selection criteria, the IF NO RECORDS FOUND clause causes the processing loop to be executed once with an "empty" record. If this is not desired, specify the statement ESCAPE BOTTOM within the IF NO RECORDS FOUND clause.

If one or more statements are specified with the IF NO RECORDS FOUND clause, the statements are executed immediately before the processing loop is entered. If no statements are to be executed before entering the loop, the keyword ENTER must be used.

Note:
If the result set of the SELECT statement consists of a single row of NULL values, the IF NO RECORDS FOUND clause is not executed. This could occur if the "selection" list consists solely of one of the "aggregate-functions" SUM, AVG, MIN or MAX on columns, and the set on which these "aggregate-functions" operate is empty.
When you use these "aggregate-functions" in the above-mentioned way, you should therefore check the values of the corresponding null-indicator fields instead of using an IF NO RECORDS FOUND clause.

Database Values

Unless other value assignments are made in the statements accompanying an IF NO RECORDS FOUND clause, Natural resets to empty all database fields which reference the file specified in the current loop.

Evaluation of System Functions

Natural system functions are evaluated once for the empty record that is created for processing as a result of the IF NO RECORDS FOUND clause.

Join Queries

A join is a query in which data is retrieved from more than one table. All the tables involved must be specified in the FROM clause.

Example:

```
DEFINE DATA LOCAL
1 #NAME     (A20)
1 #MONEY    (I4)
END-DEFINE
...
SELECT NAME, ACCOUNT
  INTO #NAME, #MONEY
  FROM  SQL-PERSONNEL P, SQL-FINANCE F
  WHERE P.PERSNR = F.PERSNR
    AND F.ACCOUNT > 10000
    ...
```

A join always forms the Cartesian product of the tables listed in the FROM clause and later eliminates from this Cartesian product table all the rows that do not satisfy the join condition specified in the WHERE clause.

Correlation-names can be used to save writing if table names are rather long. Correlation-names must be used when a column specified in the selection list exists in more than one of the tables to be joined in order to know which of the identically named columns to select.

